January 2001

PennAspect: Two-Way Aspect Model Implementation

Andrew I. Schein
University of Pennsylvania

Alexandrin Popescul
University of Pennsylvania

Lyle H. Ungar
University of Pennsylvania, ungar@cis.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-01-25.

PennAspect: Two-Way Aspect Model Implementation

Abstract

The two-way aspect model is a latent class statistical mixture model for performing *soft* clustering of co-occurrence data observations. It acts on data such as document/word pairs (words occurring in documents) or movie/people pairs (people see certain movies) to produce their joint distribution estimate. This document describes our software immplementation of the aspect model available under GNU Public License (included with the distribution). We call this package **PennAspect**. The distribution is packaged as Java source and class files. The software comes with no guarantees of any kind. We welcome user feedback and comments. To download PennAspect, visit: http://www.cis.upenn.edu/datamining/software_dist/PennAspect/index.html.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-01-25.

PennAspect: Two-Way Aspect Model Implementation *

Andrew I. Schein†, Alexandrin Popescul, Lyle H. Ungar
{ais,popescul,ungar}@seas.upenn.edu

Abstract

The two-way aspect model is a latent class statistical mixture model for performing *soft* clustering of co-occurrence data observations. It acts on data such as document/word pairs (words occurring in documents) or movie/people pairs (people see certain movies) to produce their joint distribution estimate. This document describes our software implementation of the aspect model available under GNU Public License (included with the distribution). We call this package **PennAspect**. The distribution is packaged as Java source and class files. The software comes with no guarantees of any kind. We welcome user feedback and comments. To download PennAspect, visit:
http://www.cis.upenn.edu/datamining/software_dist/PennAspect/index.html.

*University of Pennsylvania Department of Computer and Information Science Technical Report MS-CIS-01-25.
†Corresponding Author.

1 A (Very Brief) Introduction to the Two-Way Aspect Model

The term *aspect model* was first used by Hofmann and Puzicha [4]. However, the set of models denoted by the term aspect model has been developed and re-developed previously in different contexts. In the contingency table literature the two-way aspect model corresponds to a specific instance of modified path models, structural equation models (SEMs), and log-linear models with latent variables. A general overview provided in [2] describes how log-linear models with latent variables are used in social sciences. Another embodiment of the aspect model is the *aggregate Markov model* of Saul and Pereira [6] applied in natural language processing. The earliest equivalent work we have found is Goodman's 1974 paper on the analysis of contingency tables [1]. Goodman derives an EM-like algorithm that closely resembles that of Hofmann and Puzicha [4]. We use the term "EM-like" to describe Goodman's algorithm because the term EM was introduced after his work.

For the sake of notational consistency with the latest work we discuss the model using the word/document co-occurrence data application (as in [4]). In this context, documents $d \in D = \{d_1, d_2, \ldots, d_N\}$, together with the words $w \in W = \{w_1, ..., w_M\}$ that they contain, form observations (d, w), which are associated with one of the latent variables $z \in Z = \{z_1, ..., z_K\}$. Conceptually, the latent variables are topics in the documents. In generative terms, one can think of a process where documents generate or "induce" the topics or latent classes, which in its turn produce words according to class specific distributions. Documents are assumed independent of words, given the topics. The joint probability distribution over documents, topics, and words is

$$\mathsf{P}(d, w, z) = \mathsf{P}(d)\mathsf{P}(z|d)\mathsf{P}(w|z).$$

Since z is hidden we sum over the possible values when calculating the joint distribution over documents and words:

$$\mathsf{P}(d, w) = \mathsf{P}(d) \sum_z \mathsf{P}(z|d)\mathsf{P}(w|z).$$

The formula above is the *asymmetric* formulation of the aspect model. The formula is asymmetric since the document is generated first and influences (indirectly through z) the word generated. Using Bayes rule, we can rewrite the distribution in its *symmetric* formulation:

$$\mathsf{P}(d, w) = \sum_z \mathsf{P}(z)\mathsf{P}(d|z)\mathsf{P}(w|z).$$

Figure 1 shows graphical notation of both the asymmetric and symmetric aspect models.

Model parameters are learned using EM (or Annealed EM) finding a local maximum of the log-likelihood of the training data. Many clustering algorithms such as K-means assume *a priori* that objects belong to a single class. In contrast, the aspect model assumes observations $(d, w)_i$ belong to a single class, but no such assumption is made about the documents. The aspect model is convenient if we believe that the latent classes represent topics, and that documents can fall under several topics. For a more detailed treatment of the model and additional applications the reader is encouraged to refer to the original sources. For our applications in recommender systems and extensions see [5] and [7].

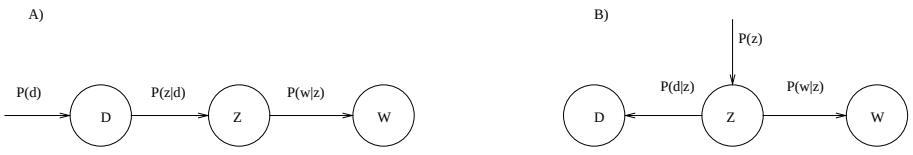

Figure 1: Graphical representation of the aspect model in A) asymmetric and B) symmetric formulations.

2 An Overview of Our Implementation

Our implementation closely follows the model and algorithm exposition given in Hofmann's Probabilistic Latent Semantic Analysis (PLSA) paper [3].

2.1 External Libraries

Our implementation relies on three third-party libraries, available under separate licenses. These libraries can be downloaded for free from their respective web sites. The first is a (sparse) matrix package from Ops Research. We use sorting routines from the Cern Colt package. The java Arguments-1.0 handles command-line parsing. Links to the packages are at our download site:
`http://www.cis.upenn.edu/datamining/software_dist/PennAspect/index.html`

2.2 Training

Train the model by invoking the PennAspect class with the appropriate command line arguments. Here is an example:

```
java -Xmx450M ungarlab.genmodels.aspect2.PennAspect --trainfile=./.data/TRAINACTOR
--valfile=./.data/VALIDATEACTOR --classes=9 --numiterations=100 --numrestarts=3
--parameters=./param
```

where `numrestarts` and `parameters` are not required (see defaults later in this section).

We tell the PennAspect class where the training and validation files are located, the number of classes, and the number of iterations to train for. If PennAspect detects overfitting (overfitting here is degrading likelihood) on the validation data before `numiterations`, it will stop. Think of the `numiterations` parameter as the maximum number of allowed iterations. The `numrestarts` parameter determines the number of times to start over with different random seeds. Recall that the EM algorithm converges to a local maximum of the likelihood surface. By restarting multiple times, we hope to find a better model fit. The `parameters` parameter tells PennAspect where to write out the parameters. PennAspect writes files with names `p_d_z`, `p_w_z` and `p_z` in this directory. Additional parameters are: `[--betastart=num] [--betastop=num] [--anealcue=num] [--betaModifier=num]`. `betastart` is the initial beta value for annealed EM, we stop training if β reaches `betastop`. We use `betaModifier` to update β when overfitting occurs. The `anealcue` is the minimum allowed difference between likelihood in consecutive iterations (validation likelihood is used here). If the difference is less than anealcue, we update β.

Defaults are:

```
parameters: ./param
numiterations: 100
numrestarts: 1
betastart: 1.0
betastop: 0.6
anealcue: 0.0
betamodifier: 0.92
```

For many applications you can leave the parameters at default settings.

As you can see, the command to execute training is long-winded, and so we recommend creating a shell script wrapper.

2.3 File Formats

2.3.1 Training/Validation Format

The Aspect code expects two files as input: a training file and a validation file. The validation file is used to stop training before overfitting occurs. Both training and validation files share the same format. Here is an example:

```
Documents: 944 Words: 1683
1       1       1.0
1       10      1.0
1       13      2.0
1       16      1.0
.
.
.
```

The first line (the label line) gives sufficient information to initialize an empty (all-zero) sparse array. For instance, in this case we know to build a 944×1683 array. Note that this means the ids for "Documents" range between 0 and 943. The second line of data states that Document 1 contains word 10 once, and so on. In other words, the first column is the document, the second column is the word, and the third column is the event count. The third column uses real numbers because that is the internal representation of the underlying SparseMatrix package. We streamline the file format by not including zero counts: unmentioned row/column pairs are implicitly set to zero in the sparse matrix representation. The label line uses a single space to separate the fields. Tab characters separate the fields of the data lines.

Observations can occur in any order; in other words the following is also a legitimate event file:

```
Documents: 944 Words: 1683
5       100     3.0
2       10      1.0
1       10      1.0
.
.
.
```

The format for the label line is: `label1: num label2: num`. You can insert your own labels. However, PennAspect will output files with `p_d_z` and `p_w_z` without regard to what labels are used in the event files. We suggest either using the labels in the example above in order to avoid confusion or changing the code to output parameter files with more informative names.

2.4 Event Manipulation

There are some java classes included that you may find handy for building and manipulating your event files. See the class `ungarlab.util.SparseMatrixUtil` for static methods for reading and writing sparse matrices.

2.5 Parameter Output

The parameters for the aspect model consist of the probabilities $\mathsf{P}(d|z)$, $\mathsf{P}(w|z)$ and $\mathsf{P}(z)$. The file formats for storing $\mathsf{P}(w|z)$ and $\mathsf{P}(d|z)$ are identical.

The PennAspect application outputs the parameters $\mathsf{P}(d|z)$ as a file `p_d_z`:

```
Array Size: 944 9
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.002000307625777631 0.003163785053340181 0.0017949511869198194 0.00211179624957
47662 0.004051748080996417 2.4263973512587983E-4 0.0019393151787399478 0.0019695
41678265096 0.006750177361385538
.
.
.
```

Consider this file a flat-file representation of a two-dimensional array of doubles. In this case, the array should have dimensions 944×9. For your own convenience, the java classes `ungarlab.util.TwoDDoubleArrayReader` can be used to read in the parameter files.

`ungarlab.util.TwoDDoubleArrayWriter` can be used to write out an arbitrary two-dimensional array of doubles.

The parameters $\mathsf{P}(z)$ are stored in a single-dimensional array, and written out as follows:

```
Array Size: 9
0.09329648151576923 0.18808634858835915 ...
```

In other words, a label line, followed by a space-separated list. Some classes for manipulating single dimensional arrays are: `ungarlab.util.DoubleArrayReader` and `ungarlab.util.DoubleArrayWriter`.

2.6 Using the Trained Model

We include a class `ungarlab.genmodels.aspect2.PennAspectReader` to read parameter files and make predictions $\mathsf{P}(d, w)$. The model is trained using a training set and a validation set. We recommend keeping a third independent data set around for testing (assuming you want to test the model). Testing the performance of a trained aspect model is inherently application specific. See Hofmann's work in comparing querying methods [3] or our own work in benchmarking recommender systems [7].

2.7 PennAspect Output Format

Here is some sample output after training with PennAspect begins:

```
Reading SparseMatrix from ./.data/TRAINEVENTS
Reading in sparse matrix with dimensions 944 x 2769
The sparse matrix contains 272903 non-null elements
Reading SparseMatrix from ./.data/VALIDATEEVENTS
Reading in sparse matrix with dimensions 944 x 2769
The sparse matrix contains 77033 non-null elements
Going to run the model with 100 iterations.
Starting average likelihood: 6.425559698841752E-7
iteration       trainLogLikelihood      testLogLikelihood       Avg. Likelihood
0       -690502.8791312637      -148441.2543897211      4.993948863492486E-6
1       -592555.5510970562      -148321.7060004806      4.9865374398419E-6
```

PennAspect should automatically stop training when either

1. We have reached the specified maximum number of iteration
   or
2. The β temperature has fallen below `betaMin` command-line parameter. (See next section)

2.8 Annealed EM Implementation

Our implementation of Annealed EM generally follows the description in [3]. There are two rules for updating β. We update β if the average event likelihood decreases (see next section). We will also update β if the difference in test set likelihood between iterations is less than the `anealcue` command line parameter.

When β is updated, we "back up" the state of the parameters and re-do the EM step. This will be reflected in the iteration counter of the output.

2.9 Why Do We Use the Average Event Likelihood?

Our code is robust to having objects that never occurred in the training set occur validation set. For instance, if we are clustering documents and words, we may have words that don't occur in any document in the training set events. However, that word may occur in the validation set... leading to a (negative) infinite log-likelihood. Our code deals with this situation by not using events with infinite log-likelihood in the likelihood calculations; we throw these events away in the likelihood calculations. This can lead to

interesting overfitting problems where the likelihood increases because certain events that DO happen in the training set all of a sudden have probability 0. We have observed this to happen in practice quite late in the training. We use the average likelihood of an event to track this phenomenon, and prevent overfitting.

Think of the average event likelihood as a safeguard against potential numerical errors.

References

[1] Leo A. Goodman. Exploratory latent structure analysis using both identifiable and unidentifiable models. *Biometrika*, 61:215–231, 1974.

[2] Jacques A. Hagenaars. *Loglinear Models with Latent Variables.* Quantitative Applications in the Social Sciences. Sage Publications, Inc., 1993.

[3] Thomas Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 289–296, 1999.

[4] Thomas Hofmann and Jan Puzicha. Latent class models for collaborative filtering. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 688–693, 1999.

[5] Alexandrin Popescul, Lyle H. Ungar, David M. Pennock, and Steve Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-2001)*, 2001.

[6] Lawrence Saul and Fernando Pereira. Aggregate and mixed-order Markov models for statistical language processing. In Claire Cardie and Ralph Weischedel, editors, *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 81–89. Association for Computational Linguistics, Somerset, New Jersey, 1997.

[7] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Generative models for cold-start recommendations. In *Proceedings of the 2001 SIGIR Workshop on Recommender Systems.*